# MINT 719 CAPSTONE PROJECT

## XBee Based Wireless Data Relay Network and Data Logger

**Master of Science**

**In**

**Internetworking**

**Submitted By**

**Aishwarya Subramaniam**

**asubrama@ualberta.ca**

**Submitted to**

**Prof. Mike Macgregor**

**March 5, 2014**

# ACKNOWLEDGEMENT

First and foremost, I would like to express my deepest gratitude to my supervisor, Prof. Mike Macgregor, for his invaluable advice, motivation and guidance throughout my graduate studies and especially the CAPSTONE project at University of Alberta. His unconditional support and constant encouragement were vital for me, without which, this project and my graduate studies would not have been possible. I learnt a great deal, both as a student and as person, while working under his able supervision.

I sincerely thank all my fellow students MINT for the wonderful discussions and inputs, making my student life a memorable one. Furthermore, I would like to extend my heartfelt gratitude to all my professors for all the knowledge shared, help and patience during the courses at University of Alberta. Also, a big hug to all my friends I met during the past 2 years, especially Vinoth, Vinay, Gomathi, and Kavin for their unconditional friendship and memorable times.
Finally and most importantly, my deepest love and gratitude is devoted to my parents and brother, who give meaning to my life. To them, I dedicate this project.

# ABSTRACT

"XBee based wireless data relay network and data logger" is an advanced concept of XBee module usage to transmit multiple node data to centralized data logging station. It facilitates to route data packets through the data router station to data logging station from multiple sensor nodes. Apart from that data packet validation and data storage in SD card is capable. The full system consists of three types of units. Each has its own operation behaviors and capabilities. Sensor device is the end device in the system and it can monitor the temperature and humidity of the environment. Then it composes a specific data packet including the device ID and measured values. Then it sends to the data router station over the wireless communication using XBee modules. Data router station is always listening to its cluster of sensors and collect data from them. After receiving some data from a node it does basic packet structure validation and routs the data packet to logger station only if the validation is passed.

Data logger station listens to the data router station and capture packets from it. Secondary data packet validation is taken place once data is received from the data router. If the validation is passed, the data is stored in a readable format in the SD card including the original sensor node ID, Temperature and humidity value. However the invalid data packets are simply discarded and wait for the next valid data packet on both router and data logger stations.

Authentication is taken place through the MAC address validation of each device. There can be lot of end sensor modules in the same environment, but the routing path and end data logger station can be vary depending on their configuration. High power XBee wireless transceivers support for long range data communication and the typical coverage can be approximately doubled since the message is routed through the data routing station. So that this system is specially designed for long rang of data communication is necessary.

## List of Figures

# Table of Contents

# Chapter 1

## Introduction

XBee based wireless data relay network and data logger consists of three basic types of units. All these units together perform a cluster sensor network with a centralized data logger.

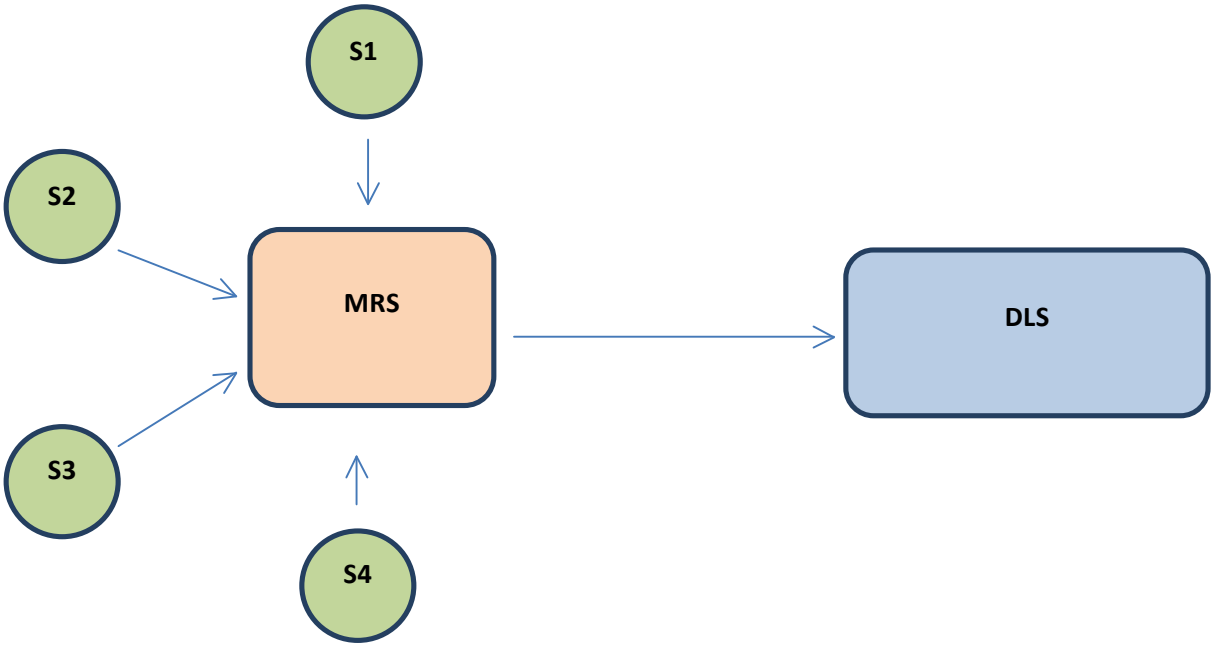


**Fig 1.1 Wireless Sensor Units**

- **S1, S2, S3, S4… = Wireless sensor units**
- **MRS = Message Relay station**
- **DLS = Data logger station**

### 1.1 Wireless sensor units

Wireless sensor units consists of two basic modules.

- Digital temperature and humidity sensor
- XBee based wireless transceiver module

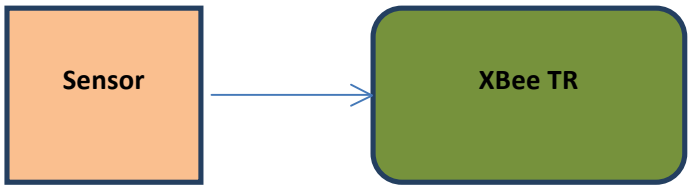


**Fig 1.2 Temperature/Humidity Sensor with wireless transceiver module**

- **Sensor** – Temperature and humidity sensor (SHT11 sensor module)
- **XBee TR** - XBee wireless transceiver

**SHT11 Sensor hardware specification**

- Temperature range: -40 °F (-40 °C) to +254.9 °F (+123.8 °C)
- Temp. accuracy: +/- 0.5 °C @ 25 °C
- Humidity range: 0 to 100% RH
- Absolute RH accuracy: +/- 3.5% RH
- Low power consumption (typically 30 µW)

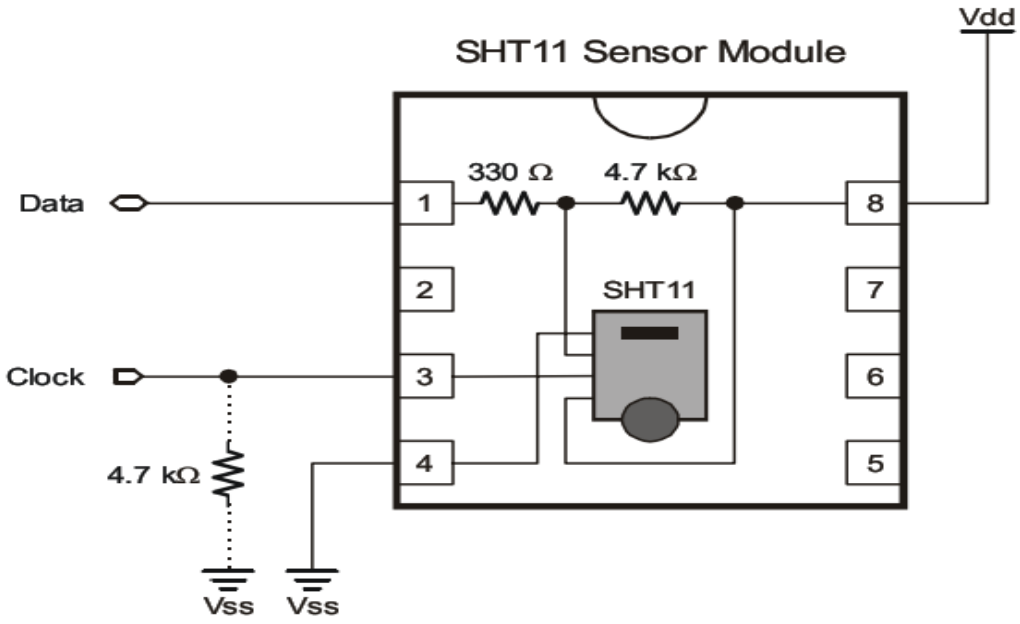


**Fig 1.3 Sensor hardware specification**

### 1.2 Data relay station

The relay station consists of two basic modules and it can do basic two operations.

- Arduino based controller module
- XBee wireless transceiver module

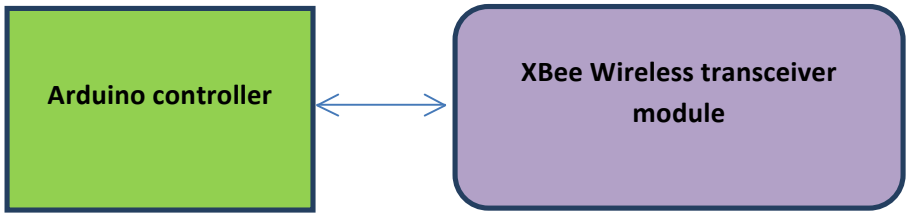


**Fig 1.4 Relay Station**

- **Basic operations**
  - **Act as a temperature and humidity monitor.**

    The SHT11 temperature and humidity sensor can directly be connected to this unit. Hence, this relay station can be used as other sensor station as well. This option can be used in case of only one sensor requirements.

  - **Act as a message relay station of other sensor nodes.**

    The default operation of this relay station is to relay messages to data logger station which comes from cluster of sensor nodes.

### 1.3 Data logger station

The data logger station consists of three basic modules. This is the all sensor nodes data logger station in the system. The SD card keeps all the information about the wireless sensor node cluster including the node ID.

**Fig 1.5 Data logger station**

- **Wireless unit** – XBee wireless transceiver module.
- **Arduino controller** – This is the controller board which organizes the wirelessly received data from all sensor stations and store in the SD card. All the received data is been validated and make error free information before saving to the SD card.
- **SD card adaptor** – FAT32 formatted micro SD card can be used with this module to store the data measured by sensors.

# Chapter 2

## Hardware specification

### 2.1 Arduino Uno

Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. The Arduino Uno is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller.

This is the core processor board of each unit. The hardware and specification can be found below.

- Microcontroller - ATmega328P-PU
- Operating Voltage – 5v
- Input Voltage (recommended) – 7-12V
- Input Voltage (limits) – 6-20V
- Digital I/O Pins - 14 (of which 6 provide PWM output)
- Analog Input Pins – 6
- DC Current per I/O Pin- 40mA
- DC Current for 3.3V Pin – 50mA
- Flash Memory - 32 KB (ATmega328) of which 0.5 KB used by bootloader
- SRAM - 2 KB (ATmega328)
- EEPROM - 1 KB (ATmega328)
- Clock Speed - 16 MHz

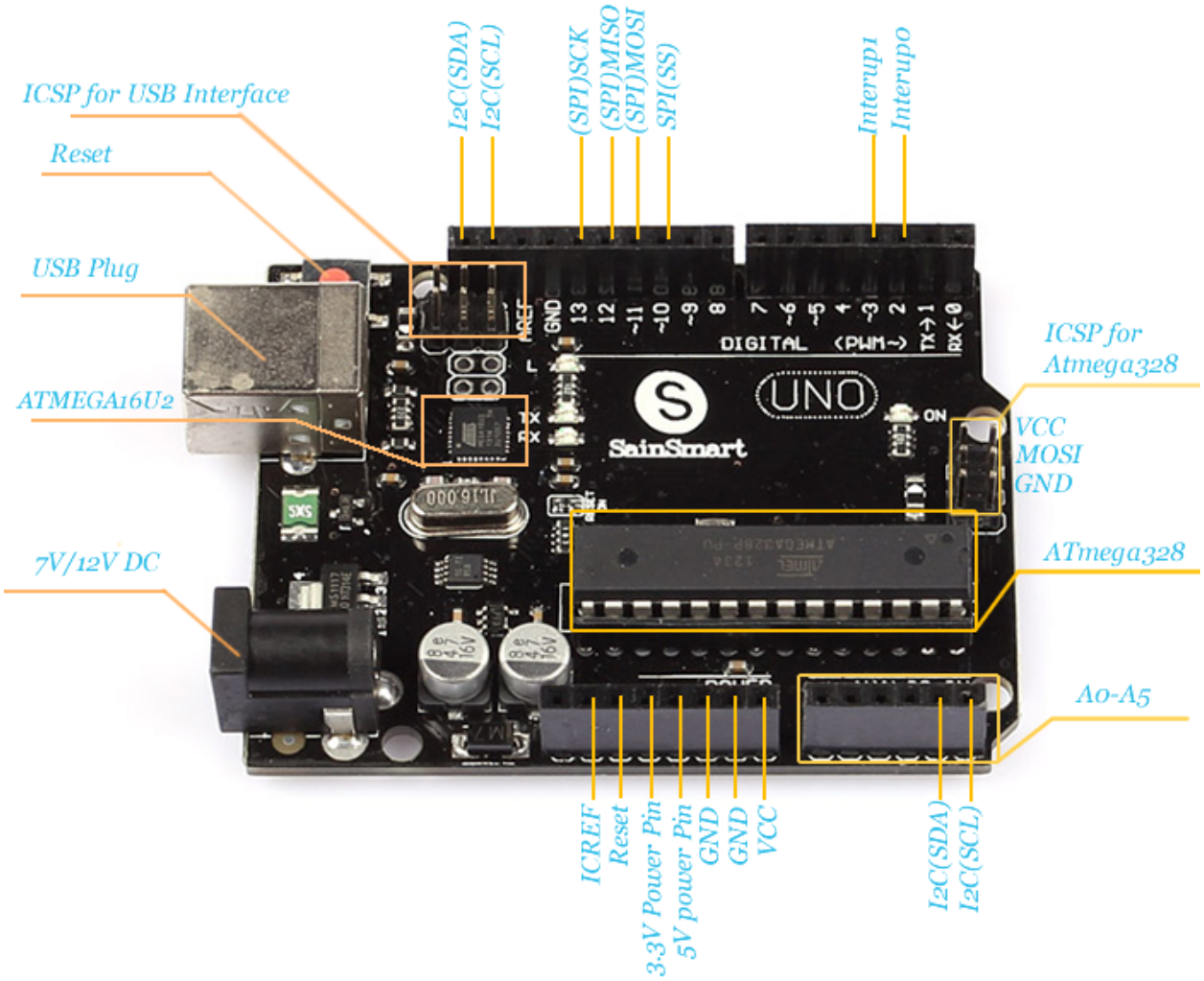


**Fig 2.1 Arduino Uno hardware specification**

## 2.2 XBee wireless module

XBee and XBee-PRO 802.15.4 OEM RF modules are embedded solutions providing wireless end-point connectivity to devices. These modules use the IEEE 802.15.4 networking protocol for fast point-to-multipoint or peer-to-peer networking. They are designed for high-throughput applications requiring low latency and predictable communication timing. This is the RF communication link among all the sensor modules, data relay station and data logger station.

**XBee Series 1**

| | |
|---|---|
| Indoor/Urban range | up to 100 ft. (30m) |
| Outdoor RF line-of-sight range | up to 300 ft. (100m) |
| Transmit Power Output | 1 mW (0dbm) |
| RF Data Rate | 250 Kbps |
| Receiver Sensitivity | -92dbm (1% PER) |
| Supply Voltage | 2.8 - 3.4 V |
| Transmit Current (typical) | 45 mA (@ 3.3 V) |
| Idle/Receive Current (typical) | 50 mA (@ 3.3 V) |
| Power-down Current | 10 uA |
| Frequency | ISM 2.4 GHz |
| Dimensions | 0.0960" x 1.087" |
| Operating Temperature | -40 to 85 C |
| Antenna Options | PCB, Integrated Whip, U.FL, RPSMA |
| Network Topologies | Point to point, Star, Mesh (with DigiMesh firmware) |
| Number of Channels | 16 Direct Sequence Channels |
| Filtration Options | PAN ID, Channel & Source/Destination |


**Fig 2.2 XBee-PRO S1**

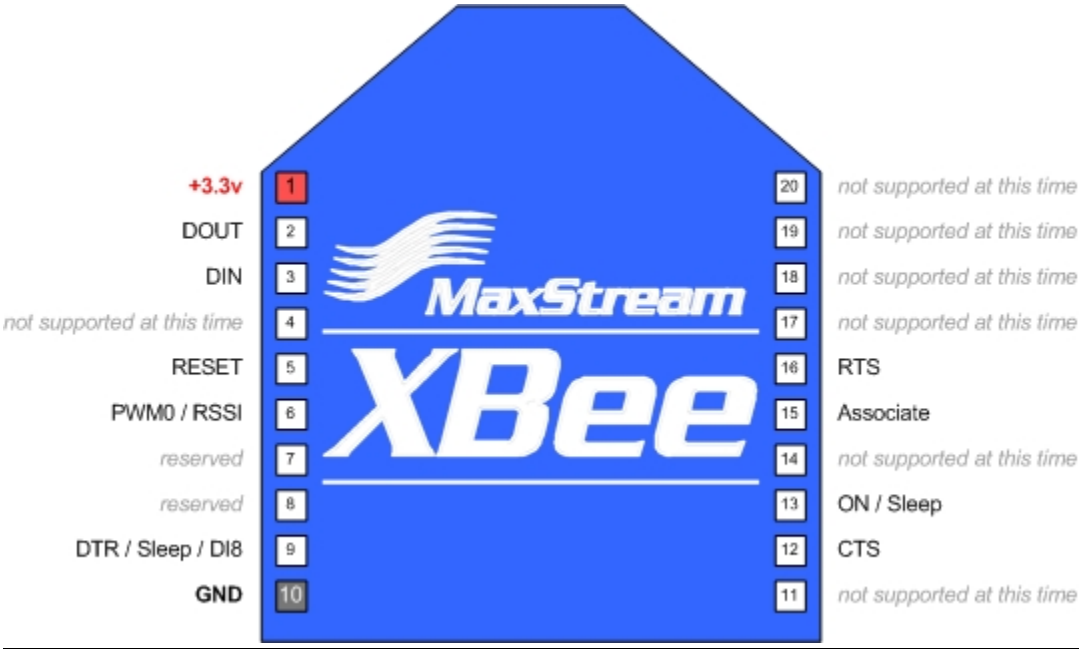


**Fig 2.3 XBee specification**

## 2.3 XBee shield

This is a unique shield for the Arduino platform. These board connects directly with the Arduino USB board and allows the Arduino to wireless communicate over a modified ZigBee protocol using the popular XBee module from max-stream. The radio is removable from the shield, so any correctly configured XBee module can be connected with the shield.

**Specifications**

- Mounts directly onto your Arduino
- DIN and DOUT pins of XBee can be connected to either the UART pins or any digital pin on the Arduino (D2 and D3 default)
- 3.3V power regulation and MOSFET level shifting on-board
- 9x11 grid of 0.1" spaced prototyping holes
- Reset button brought out to shield
- Power, DIN, DOUT, RSSI and DIO5 indicator LEDs

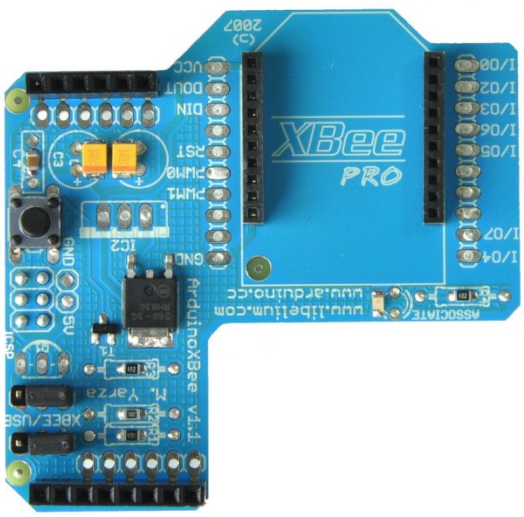


**Fig 2.4 XBee Shield hardware specification**

### 2.4 SD card module

The SD card module allows Arduino UNO to read and write Standard SD, SDHC and Micro SD cards up to 16GB using the SPI port. The SD/micro SD card can be unplugged from the board and directly connected to PC to retrieve stored data on the SD card.

**Specifications**

| Item | Min | Typical | Max | Unit |
|---|---|---|---|---|
| Voltage | 3.5 | 5.0 | 5.5 | V |
| Current | 0.159 | 100 | 200 | mA |
| Supported Card Type | SD (<=16G); Micro SD (<=2G); SDHC (<=16G) | | | - |
| Dimension | 68.7 x 53.5 x 19.00 | | | mm |
| Net Weight | 14.8 | | | g |

**Fig 2.5 SD card module specification**

## 2.5 XBee operating modes

XBee module can be configured to operate in three modes. Each mode has its own characteristics and behaviors. Depending on the requirement, the suitable mode need to be selected and the device should configure accordingly.

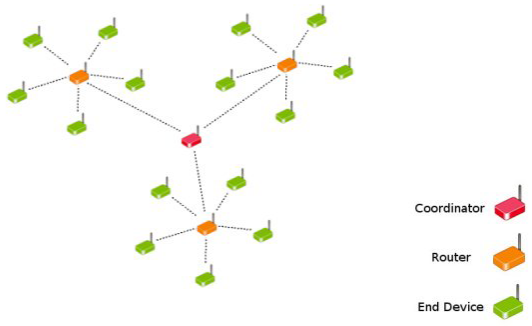


**Fig 2.6 XBee modes**

### 2.5.1 End Device mode

The end device contains enough functionality to talk to the parent node (either the coordinator or a router). It cannot relay data from other devices. This relationship allows the node to be asleep a significant amount of the time thereby giving long battery life.
This mode is applicable for sensor units in this system. It only communicates with its router and hands over the message only to the router unit (Data relay station).

### 2.5.2 Router mode

For running an application function, a router can act as an intermediate router, passing on data from other devices. This is similar as the operation of data relay station. It hands over the received data from all the sensor modules to coordinator device. (Data logger station)

### 2.5.3 Coordinator mode

The most capable device, the coordinator forms the root of the network tree and might bridge to other networks. There is exactly one ZigBee coordinator in each network as it is the device that started the network originally. Coordinator mode is similar as the data logger operation mode in the system. It only communicates with router units and accepts data from them.

## 2.6 Design and implementation

### 2.6.1 Data relay station and data logger station

X-CTU configuration software can be used to configure the XBee as required. Basic configuration must be made on XBee modules before connecting to the controller boards. Firmware versions and certain other settings can still be changed depending on the XBee modules going to be used with the application. XBee version 2 modules have been used for the following sample configurations with X-CTU version 5.2.7.5

1) Connect the XBee module to PC through the USB interface using XBee usb explorer interface board as follows. (Other than the XBee USB explorer board, any usb to serial converter module can be used to connect XBee with PC. However the operating voltage of XBee is 3.3v and it should never connect to 5V supply for any reason)
    - Arduino board itself can be used instead of the XBee usb explorer board by removing the microcontroller from the board.
    - Some advanced XBee explorer boards might be required with dedicated features for XBee hardware, to recover the damaged XBee firmware, since the conventional USB to serial boards are still not support for some resetting methods.
    - If something goes wrong while uploading the firmware, the XBee might get damage permanently. It might require special care to get it back to normal. Simple way is short circuit the ground pin (10) and reset pin (5) for a few seconds. This might help to reset the device in most of the firmware troubles.

**Fig 2.7 XBee Hardware**

2) Open the X-CTU application and select the relevant serial interface from the list. In the following example Arduino board is the relevant serial interface which connected to XBee module. Other settings can be used as it is except the Baud (Default Baud rate is 9600 and it can be configured from the X-CTU)

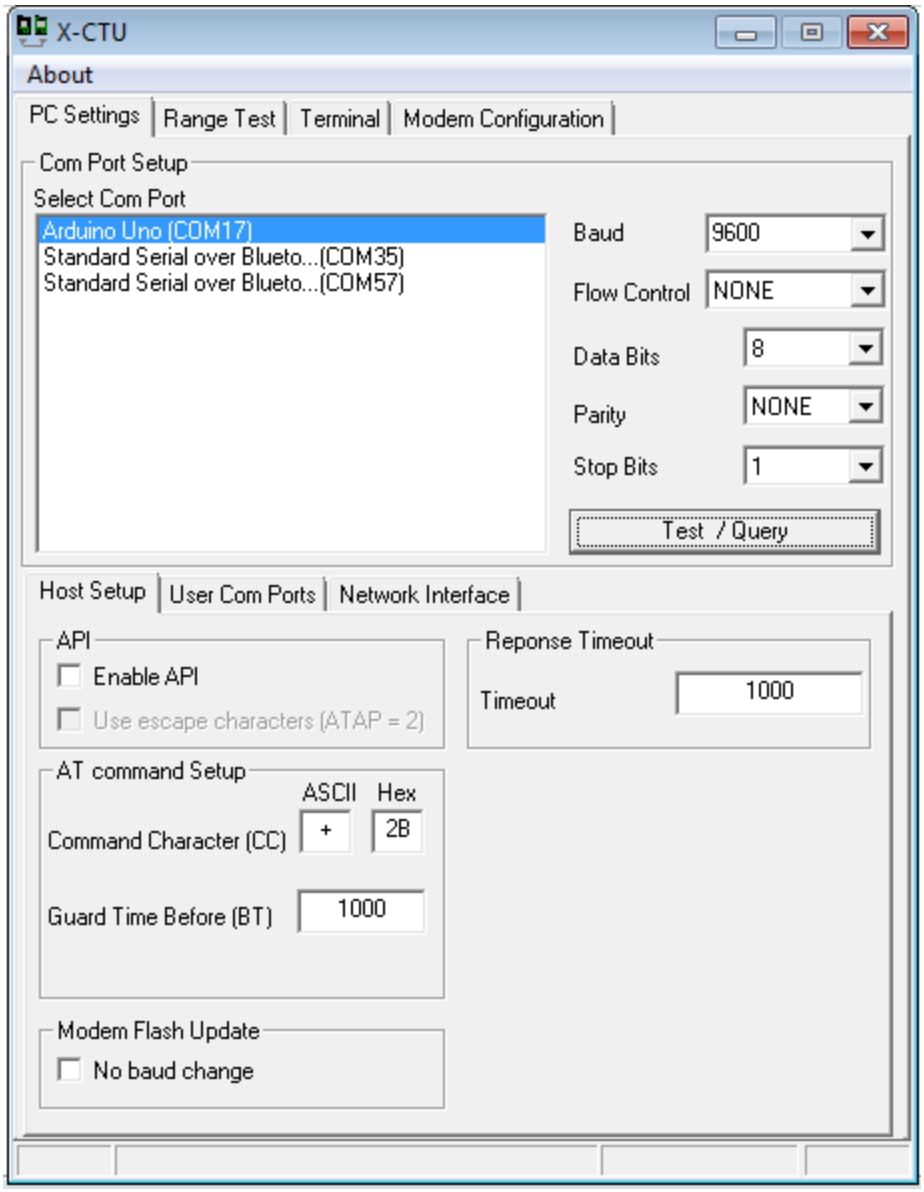


**Fig 2.8 X-CTU Settings**

3) Click on "Test/Query" button and test the connection to the module. If the connection is failed, then use different baud rate and again test the connection with the device.(Factory default baud rate of most of the XBee models is 9600)

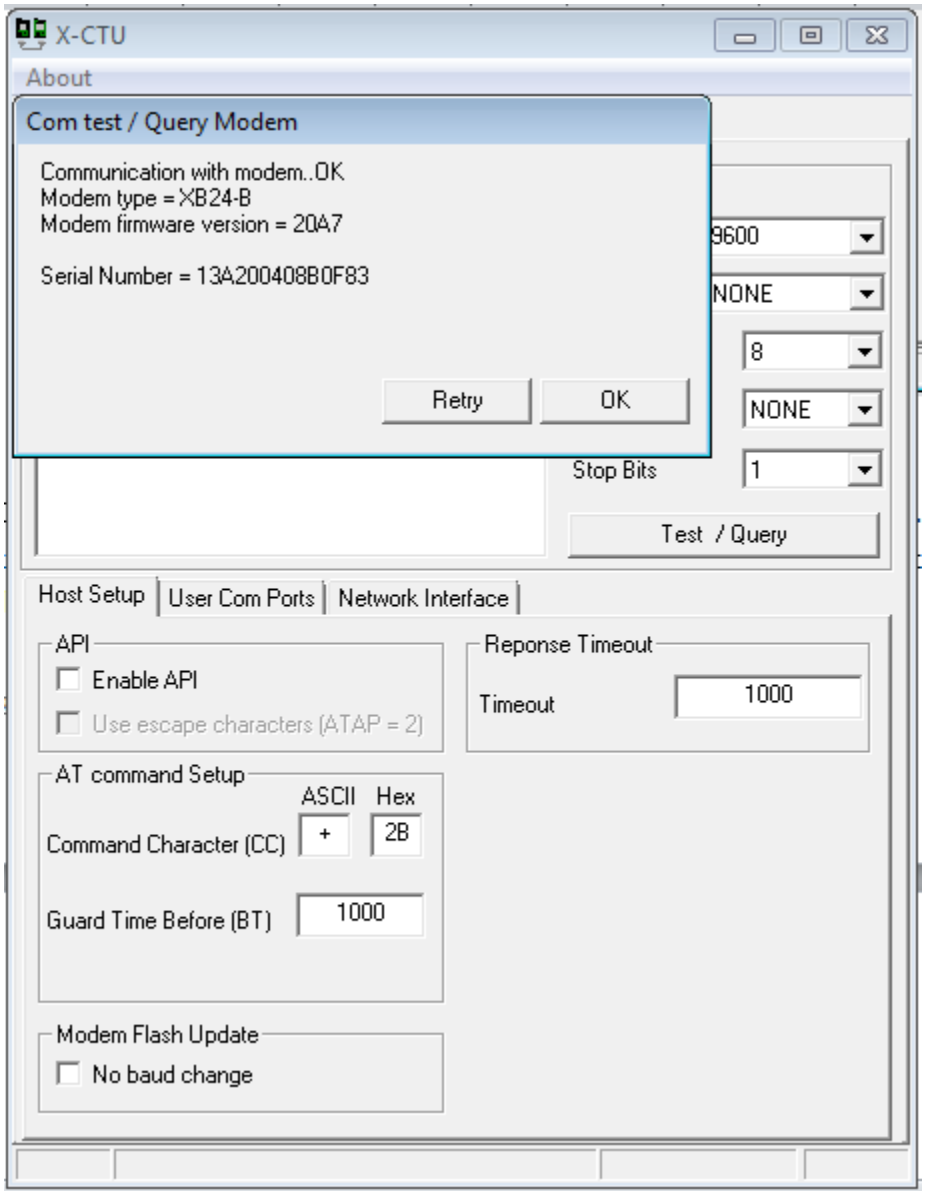


**Fig 2.9 X-CTU Test/Query**

4) Now click on "Modem Configuration" tab and make sure to check the "Always Update Firmware" check box. Then click on "Read" button. This action will read the all existing configurations from the XBee and then can be edited all relevant parameters as required. (Note down the **SH** and **SL** values, in this example they are as follows)

   # SH13A200

   # SL408B0F83

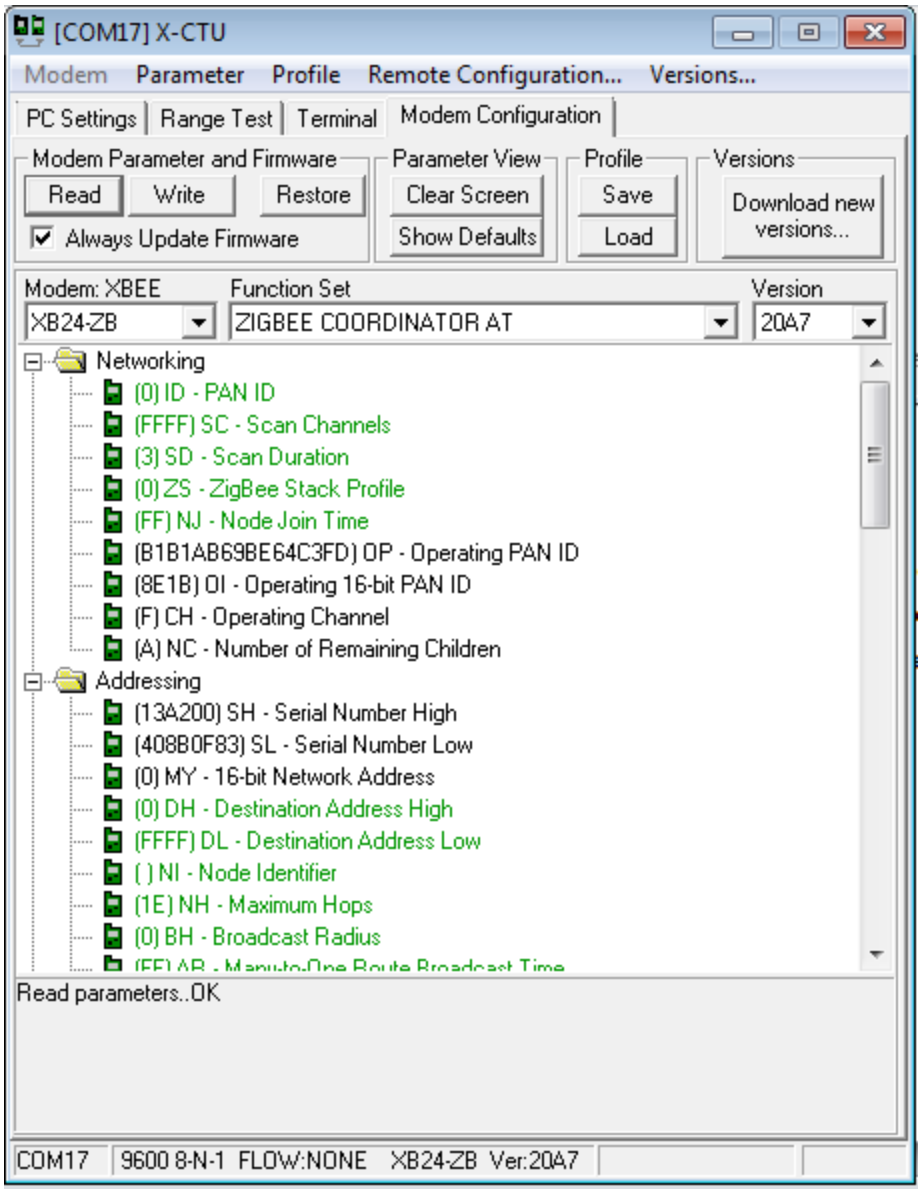


**Fig 2.10 X-CTU modem configuration**

5) Change the following settings as listed below and click on "write" button to upload the new settings to XBee module.
   - **Modem :** XB24-ZB
   - **Function set:** ZIGBEE ROUTER AT

- **PAN ID:** any 4(1234) digit number (This number must be unique among all the devices in the network)
- **Baud rate:** 57600 (option 6)

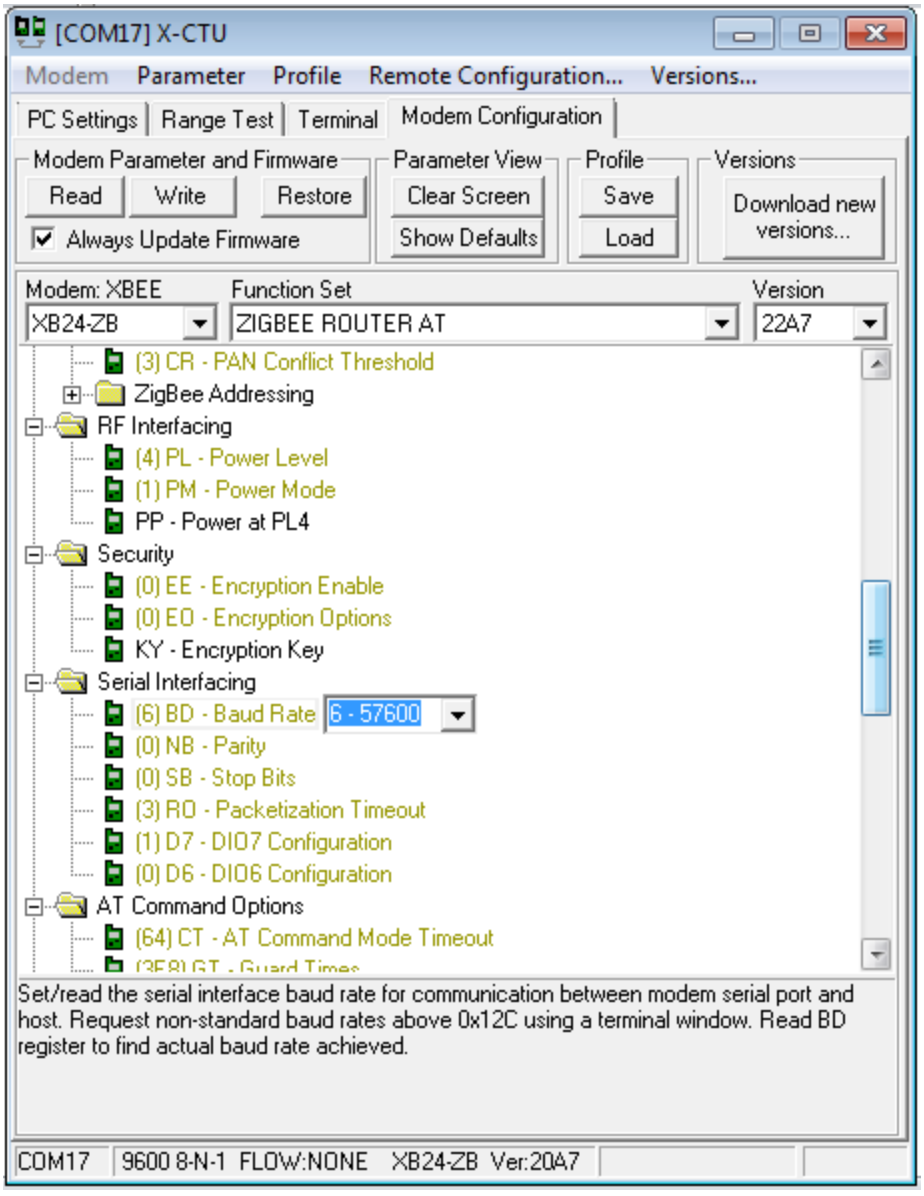


**Fig 2.11 Write version and settings**

6) Then connect the other XBee unit to the PC and open other X-CTU terminal to read the settings from other XBee. It need to insert the “SH” and “SL” code of the new XBee to previously connected one to the PC. Note down the SH and SL values of new device as follows after reading the device configurations.

# SH: 13A200

# SL: 408B0F6C

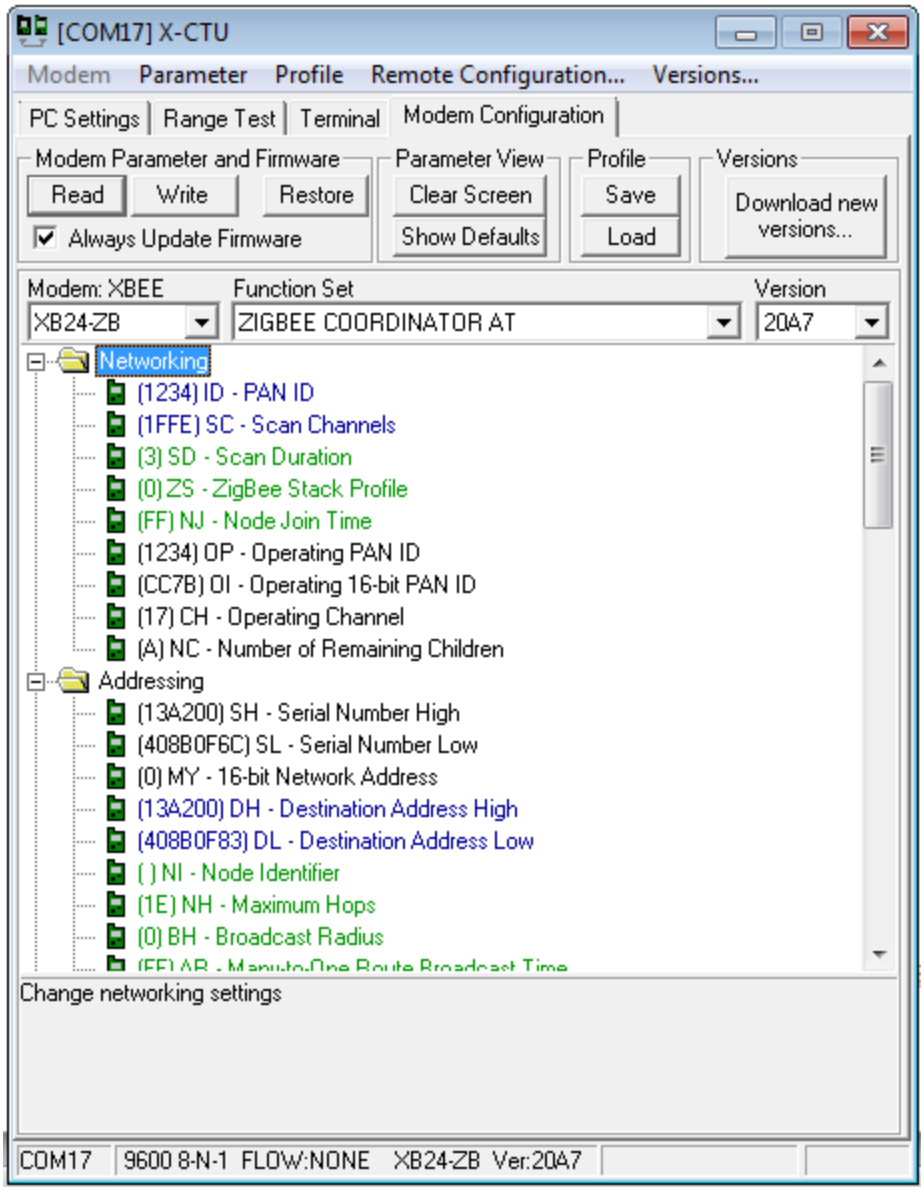


**Fig 2.12 Read version and settings**

7) Then change the following settings as listed below and click on “write” button to upload the new settings to XBee module.

- **Modem :** XB24-ZB
- **Function set:** ZIGBEE COORDINATOR AT
- **PAN ID:** any 4(1234) digit number (This number must be unique among all the devices in the network)
- **Baud rate:** 57600 (option 6)
- **DH:** 13A200 (SH of router XBee)
- **DL:** 408B0F83 (SL of router XBee)

8) As same as above step set the DH and DL values of router device from the SH and SL values of Coordinator XBee respectively. After that read the configurations, then it should looks like this.

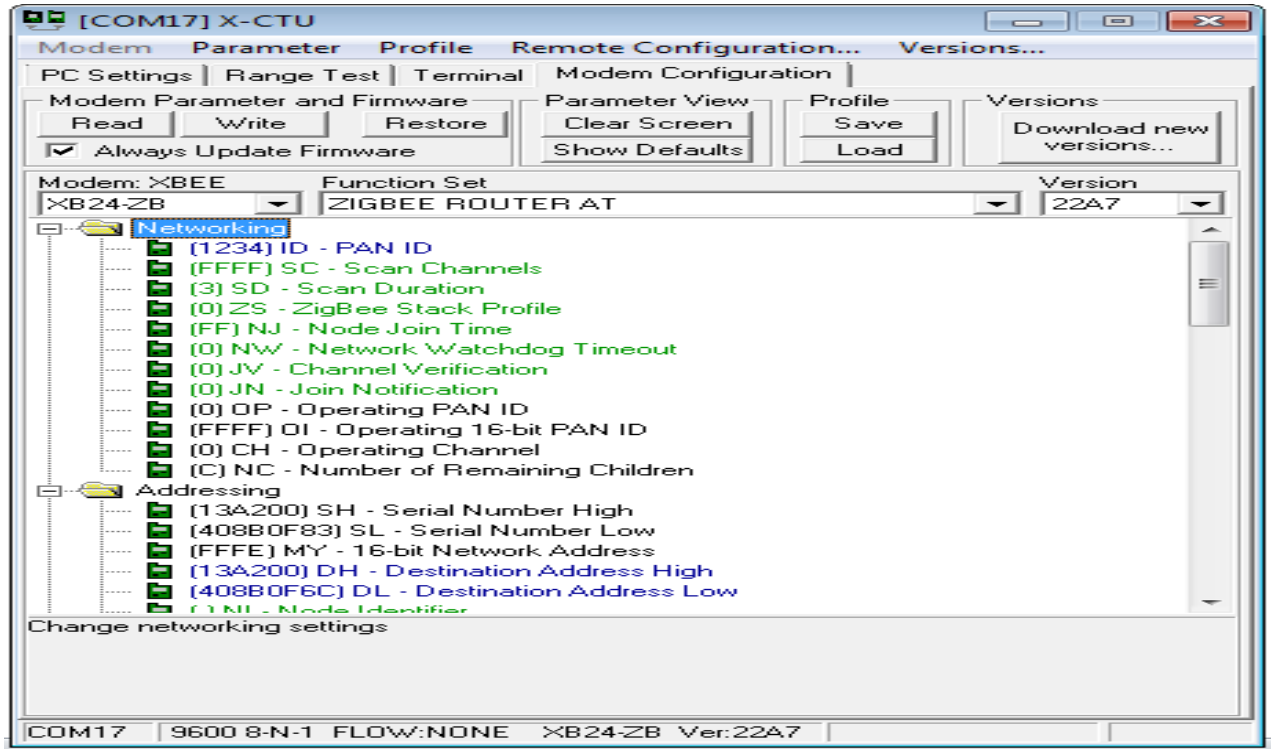


**Fig 2.13 Read configuration**

**Configuration Summary**



**ZIGBEE COORDINATOR AT**

1) PAN ID: 1234
2) SH: 13A200
3) SL: 408B0F6C
4) DH: 13A200
5) DL: 408B0F83
6) Baud rate: 57600

**ZIGBEE ROUTER AT**

1) PAN ID: 1234
2) SH: 13A200
3) SL: 408B0F83
4) DH: 13A200
5) DL: 408B0F6C
6) Baud rate: 57600

9) Make sure that the communication link is working fine using the "Terminal" tab of the X_CTU. It can be used to access the AT command mode of the XBee module and check the saved settings in the XBee. Follow the steps given below to access the command mode of XBee and execute valid command set to read settings from XBee.
   1) Type "+++" on the terminal and wait for "OK"
   2) Then the device is in the command mode and can execute valid AT commands
   3) Normally the module will automatically exit command mode after 10 seconds (depending on the CT parameter), ATCN will exit out of command mode immediately and return the module to normal operation. This command is not required and is mainly used to save time or when sending AT commands using a microcontroller or script.

4) Execute the following useful AT command set and check the output on both modules to verify the settings.
   - ATVR –Device firmware version
   - ATID – PAN ID of the device
   - ATSH – Serial number high
   - ATSL – Serial number low
   - ATDH – Destination address high
   - ATDL – Destination address low
   - ATBD – Current baud rate of the module
   - ATCN – Exit the command mode

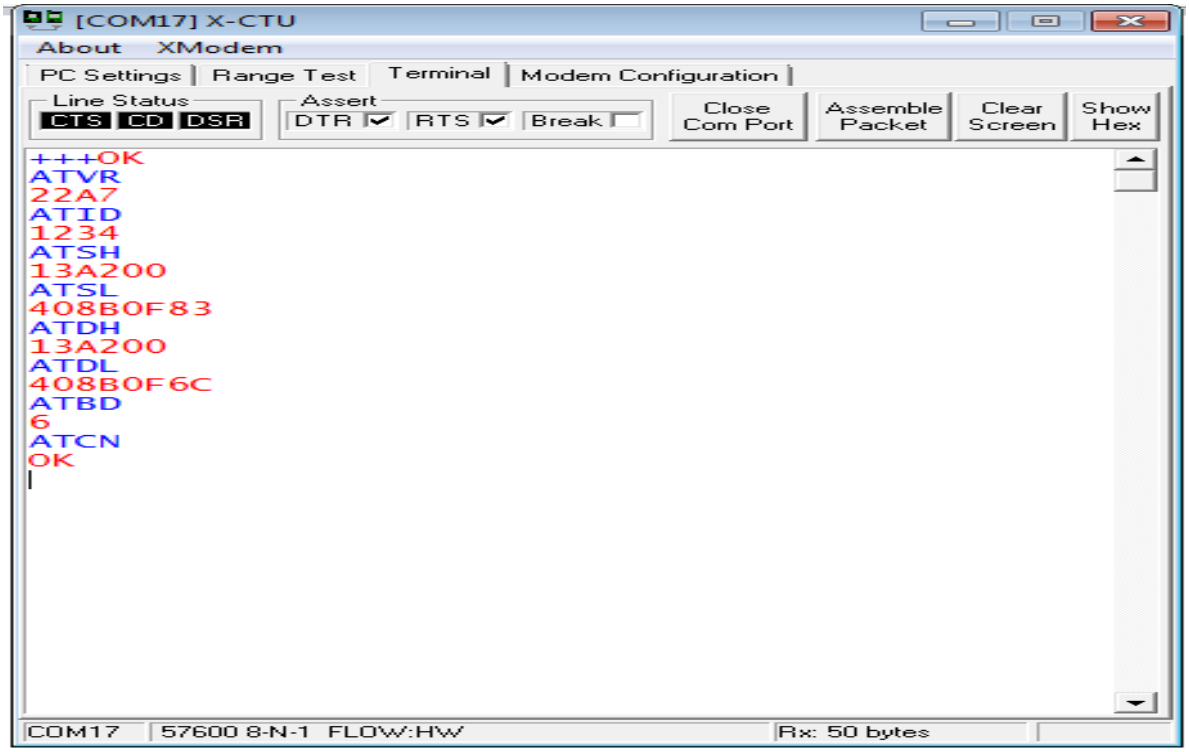


**Fig 2.14 Settings of Router module**

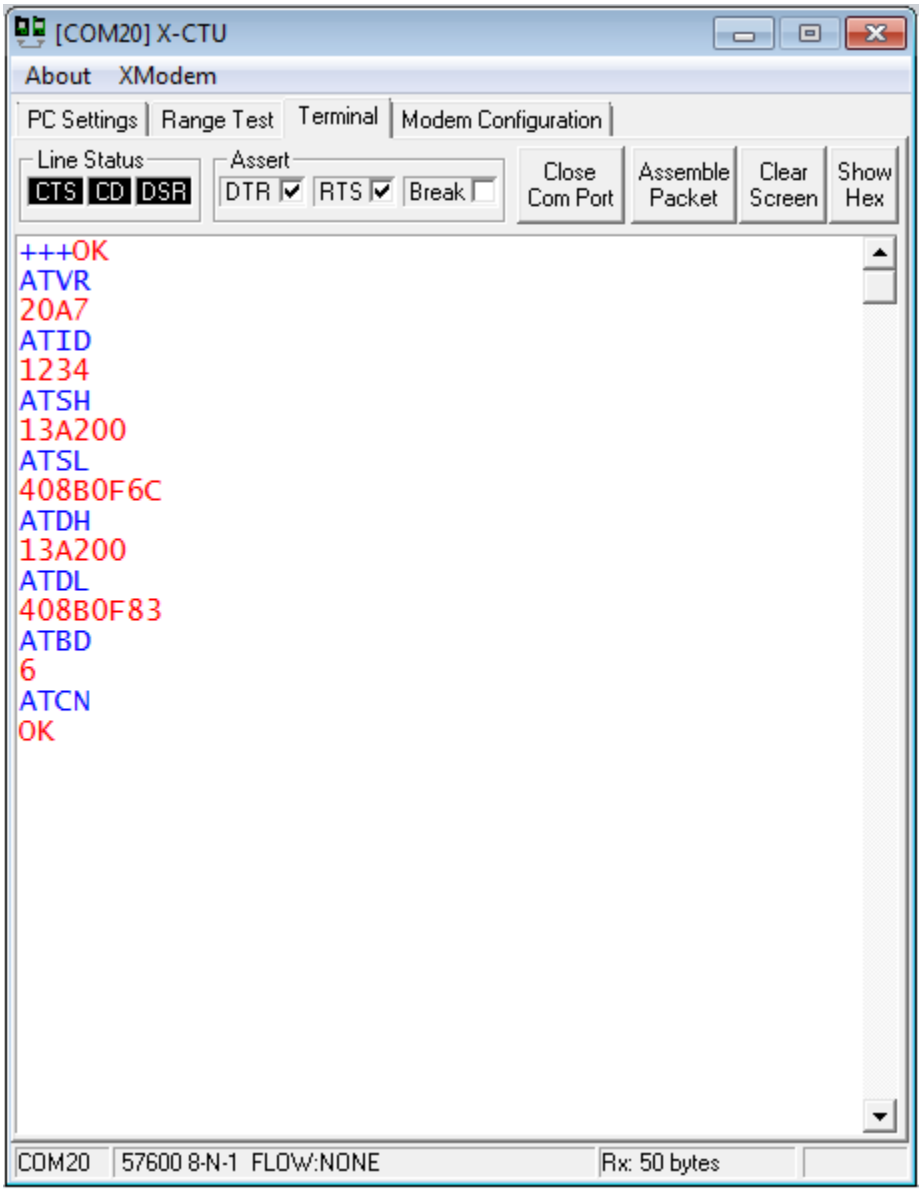


**Fig 2.15 Settings of Coordinator module**

10) After that the communication link can be tested as follows. Make sure to connect both XBee modules to PC at the same time through the XBee USB explorer boards. Choose the correct USB port and set the proper baud rate and other relevant settings. Then check the connectivity first. After that click on the "Terminal" tab on both X-CTU interfaces and type something on one terminal. It should appear on the other terminal. Then type something on other terminal and it should appear on first terminal as shown below.

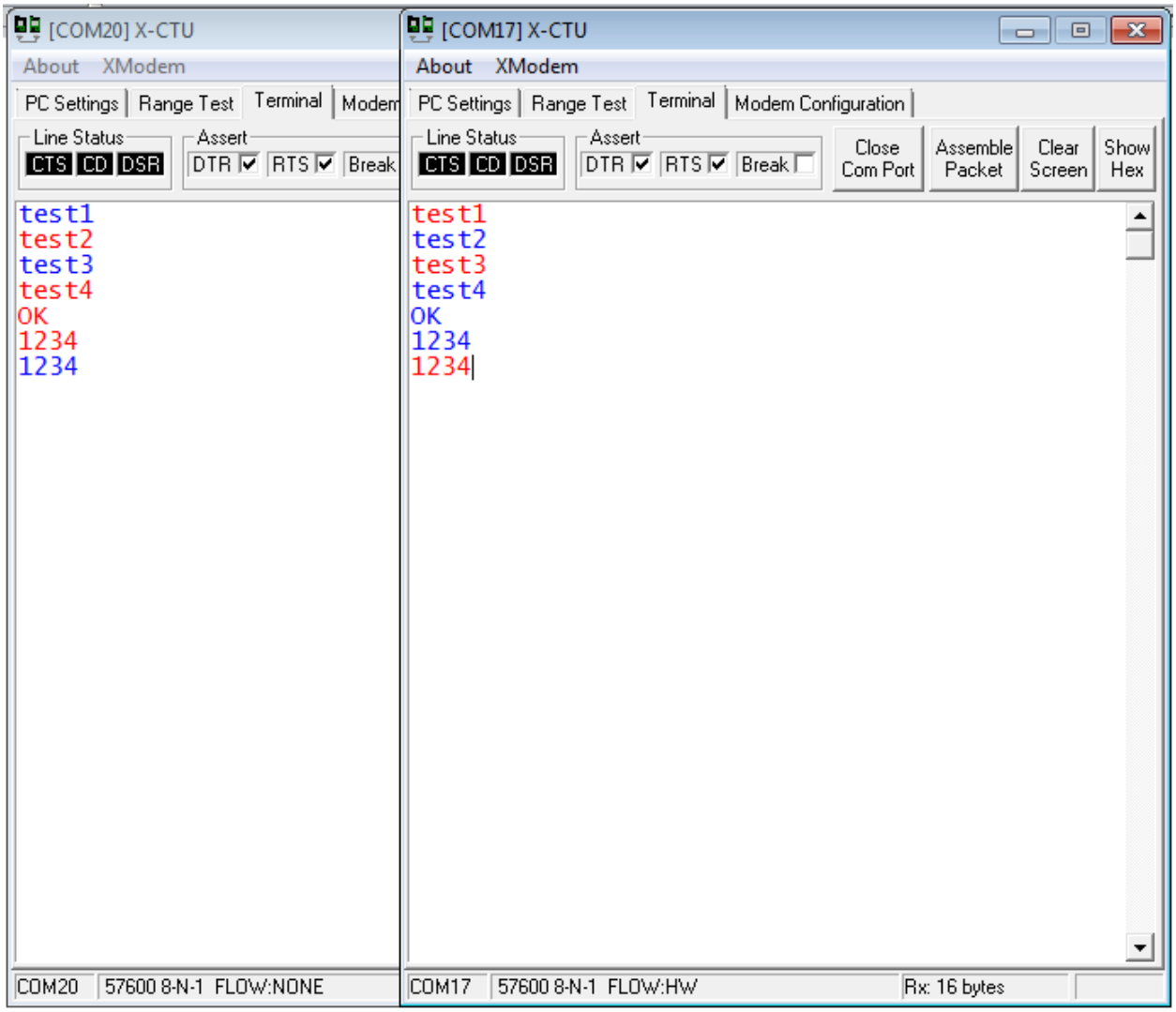


**Fig 2.16 X-CTU terminal testing**

This makes sure that the both way communication is properly established. Now the XBee modules are ready to connect to the controller boards.

### 2.6.2 Data packet structure

For the data communication among the end device (Sensor unit) to router module (Relay station) and router module to Coordinator (Data logger module) is managed using the following format of data packet. Sensor device ID is appended to the pay load structure in order to deal with multiple sensor units.

<DevID>10 T30.50C H50.40RH;

- <DevID>= Prefix string
- 10 = Device ID number
- T = Prefix character of temperature value
- 30.50 C = temperature value in Centigrade
- H = Prefix character of humidity value
- 50.40 RH = Relative humidity value

Payload structure is validated at the router device and route to the destination (Data logger unit) only the content message is error free and valid. Then the message again will be validated at the data logger module and store in the SD card only the second validation is successful.

**Router module operation can be summarized as follows,**

1) It waits for new data and does the validation if data received.
2) Discard the data packet if the validation is failed or route the message to data logger unit only if the validation is passed.
3) However the acknowledgment message is not sent at the current system to recall the data from sensors even it fails while the validation.
4) Router station can be configured to run without the validation. If it runs in that mode, it will route all the messages to the data logger module without validating the content of the message.
5) However it is highly recommended to enable the validation on the router before hand over the message to logger module. Because of avoiding the unnecessary busyness of the data logger module end.

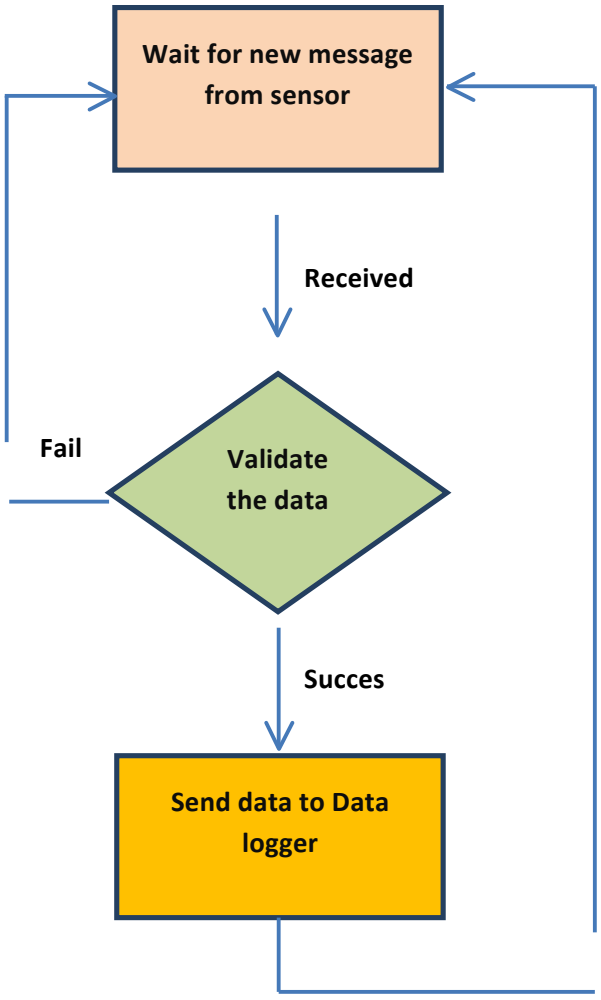


**Fig 2.17 Flowchart of Router module operation**

Routing is happening without queue the data in the router station; hence the communication is approximately real time. However the resending data to logger station will not be happen when the logger station is offline or in case of last communication is incomplete or have failed.

This type of errors can be rectified implementing following mechanisms.

1) Improve the router station code to support to resend option. This will rectify errors caused due to incomplete data communication between the router station and data logger station or sudden data link break down.

2) The XBee modules can be configured with enabling retransmit option inside the device itself. Then the XBee will be taken care of each data packet communication in its firmware level.
3) Router stations controller board has EEPROM memory in it. This can be used to store limited number of data packets temporary until successfully send them to the destination. However this will add some delay to the communication due to EEPROM read and write process is a little time consuming process.
4) Acknowledgement signal can be sent with each data communication to verify the state of last data transfer. Depending on the acknowledgement check sum value, the sender can resend the failed data or stop resending if the last communication is successful.

**Data logger module operation can be summarized as follows,**

1) It waits for new data from the router and does the validation if any data received.
2) Discard the data packet if the validation is failed or save the message to the SD card only if the validation is passed.
3) However the acknowledgment message is not sent at the current system to recall the data from router station even it fails while the validation.
4) Time stamp is not written with the original message to the SD card in the current system. However this feature can easily be enabled with additional RTC module attached to data logger station. Then all the data will be saved in the SD card with a time stamp appended to the message.
5) This will enable the system to analyze the time sensitive changesof data stored on the SD card

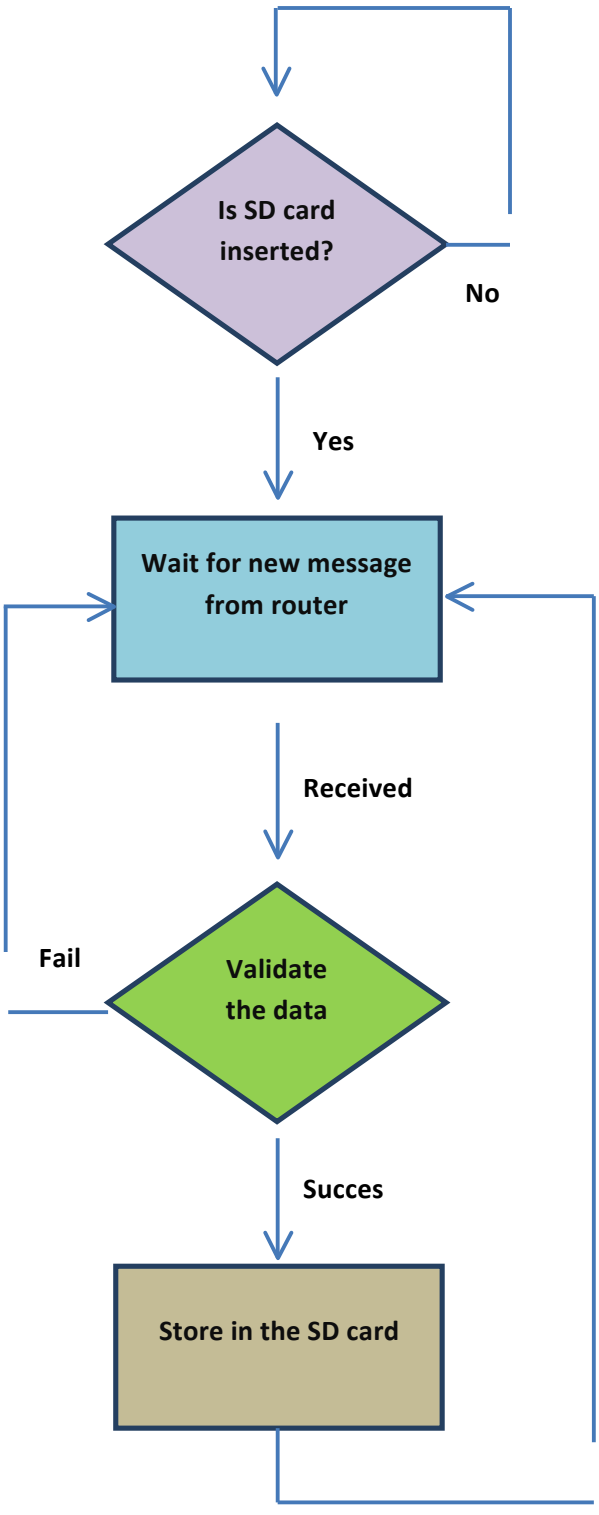


**Fig 2.18 Flowchart of data module operation**

### 2.7 Programming language and code review

The programming is handled in C++ language and standard Arduino libraries have been used in some requirements. Arduino is the development platform of each sensor station, data relay station and data logger station too.

**Standard library list used in the application**

1) SoftwareSerial.h
2) SD.h
3) string.h

**Data relay station source code**

```
**********************************************************************

#include <SoftwareSerial.h>

#include <string.h>

SoftwareSerial xbee(2, 3); // RX, TX initialize serial interface

#define PERIOD 1000       //data sending period in ms

float temprature = 30.5;   //initialize value temprature

float humidity = 50.4;     // initialize value humidity

int nodeID = 10;           // initialize node ID

char buf[10];

String message;

void setup(){

  //Serial.begin(115200);  // best speed for proper communication without any garbage characters

  //xbee.begin(115200);    // best speed for proper communication without any garbage characters
```

```
  Serial.begin(57600);// start the serial communication with baud rate 57600
  xbee.begin(57600);
}
void loop()
{
  /*
  Serial.print("N");
  Serial.print(nodeID);
  Serial.print("T");
  Serial.print(temprature);
  Serial.print("H");
  Serial.print(humidity);
  Serial.println(";");
  //Serial.println("test;");

  //xbee.print("test;");
  xbee.print("N");
  xbee.print(nodeID);
  xbee.print("T");
  xbee.print(temprature);
  xbee.print("H");
```

```
 xbee.print(humidity);
 xbee.println(";");
 delay(1000);
 */
 String str = "";
 int finish = 0;
 if (Serial.available()){
   while (Serial.available() || finish == 0) {
     char nw = (char) Serial.read();
     if(nw != '\r'){  //check for line ending character
         //Serial.println(str);
       str += nw;
     }else{
       //str += nw;
       finish = 1;
       exit;
     }
     delay(1); //wait for the next byte
   }
 }
```

```
  if(str.length()> 4){  // validate the received data
   //Serial.println(str);
   str.replace("\n", "");  // remove the new line charactor
   //Serial.println(str);
   if(str.substring(0,7)== "<DevID>"){  // search for special tag <DevID>
     int T_position = str.lastIndexOf('T');
     //Serial.println(T_position);
     if(T_position > 0 && str.substring((T_position+6),(T_position+7)) == "C" && str.substring((T_position+13),(T_position+15)) == "RH"){
       delay(1);
 Serial.println(str);  // re-send data back to data logger station.
     }
   }
   /*
   dtostrf(temprature, 4, 2, buf);  //xx.xx format conversion
   message = String("<DevID>") + nodeID +" T" + String(buf) + "C ";
   dtostrf(humidity, 4, 2, buf);    //xx.xx format conversion
   message = message + String(buf) + "RH;";
   xbee.println(message);
   Serial.println(message);
   */
```

```
  str = "";
  finish = 0;
 }
 }
```

**********

**Function description**

1) **void setup()**
   This is required function in Arduino sketcha. All the settings and variables initialized in side this function.
2) **void loop()**
   This also required function in Arduino sketcha.Basically these two functions are really necessary to run arduino sketch correctly.

No extra functions on the main code other than the essential two above functions.

This will route messages that has the following format

<DevID>10 T30.50C 50.45RH;

**Data logger station source code**

**********

```
#include <SD.h>        // include the library
#define LED 8          // define the status led connected pin

char log_file[] = "LOG.txt";   //Data log file name
double tempr = 30.50;// initialize the temperature value
double humid = 50.40;          //initialize the humidity value
```

```
//int nodeID = 10;

File myFile;
String progress;

void setup()
{
 /*
 Serial.begin(115200);
 Serial.println("++++++ xBee Data logger ++++++");
 Serial.print("Initializing SD card...");
 */
 pinMode(10, OUTPUT);
 pinMode(LED, OUTPUT);  //Status LED
 digitalWrite(LED, 0);

 if (!SD.begin(4)) {              //initialize the SD card
  progress = "Failed";
  //return;
 }else{
  progress = "Done";
 }

 if(progress == "Done"){
  //read_sd();
  write_sd("+++++++++++++++ New Session Started +++++++++++++++");
  //read_sd();
 }

 Serial.begin(57600);            //  start the serial communication
 Serial.println("++++++ xBee Data logger ++++++");
```

```
  Serial.print("Initializing SD card...");
  Serial.println(progress);

}

void write_sd(String data){              // data writing function to SD card
  myFile = SD.open(log_file, FILE_WRITE);

 // if the file opened okay, write to it:
  if (myFile) {
    Serial.println(data);          // print on serial monitor
    myFile.println(data);          // write to SD
    myFile.close();                        // close the file:
    digitalWrite(LED, 1);
    delay(5);
    digitalWrite(LED, 0);
    //Serial.println("done.");
  } else {
    Serial.println("error opening log file");      // if the file didn't open, print an error:
  }

}


void read_sd(){                  // read and print data from SD card to serial interface

  myFile = SD.open(log_file);
  if (myFile) {
    //Serial.println(log_file);

    // read from the file until there's nothing else in it:
```

```
    while (myFile.available()) {
            Serial.write(myFile.read());
    }
    // close the file:
    myFile.close();
  } else {
Serial.println("error opening test.txt");          // if the file didn't open, print an error:


  }


}
void loop(){
 if(1){
 //if(progress == "Done"){
  //Serial.println(millis());
  String str = "";
  int finish = 0;
  if (Serial.available()){          //read serial data from xbee
    while (Serial.available() || finish == 0) {
      char nw = (char) Serial.read();
      if(nw != '\r'){                  //check for line ending character
          //Serial.println(str);
        str += nw;
      }else{
        //str += nw;
        finish = 1;
        exit;
      }
      delay(1);   //wait for the next byte
    }
  }
```

```
  if(str.length()> 4){
   //Serial.println(str);
   str.replace("\n", "");            // remove the new line charactor
   write_sd(str);
   str = "";
   finish = 0;
  }
 }

}
```

**********

**Function description**

1) void write_sd()

   This is the function that writes data to SD card. One string argument is passed to this function and it is the string written to the SD card as it is.

2) void read_sd()

   In case of reading the data written to the SD card can be printed using this function. It will print all the SD card stored data to serial interface and can be seen using any serial data interface such as “putty” or “Arduino serial data reader”.

3) SD.open()

   Before doing any read, write operation, it need to open the file for reading. The file name is passed as the argument for this function. Arduino can open only one file at a time for reading or writing due to the limited memory and processing power limitations.

4) myFile.close()

   After writing data to the file, it needs to be closed properly. This function must be called after each SD.open() function call to properly close the edited file in the SD card.

### 2.8 Sample log data file

#### 2.8.1 Single node data log

Following log file shows the data file format of the SD card when only one sensor node exist in the system. The Router station operates as the sensor device in this case and no further end device is being used for monitoring. Note that the node ID of each entry is 10. That means the same device has sent all the data to the station. Further the device ID has only 2 digits. That means the data received from 2nd level device. (Router device works as a sensor)

++++++++++++++ New Session Started ++++++++++++++
<DevID>10 T30.50C 50.45RH;
<DevID>10 T30.52C 50.50RH;
<DevID>10 T30.51C 51.20RH;
<DevID>10 T30.51C 51.40RH;
<DevID>10 T30.54C 52.40RH;
<DevID>10 T30.58C 52.30RH;
<DevID>10 T30.62C 52.60RH;
<DevID>10 T30.63C 51.00RH;
<DevID>10 T30.64C 49.40RH;
<DevID>10 T30.64C 49.60RH;
<DevID>10 T30.58C 49.80RH;
<DevID>10 T30.56C 50.20RH;
<DevID>10 T30.52C 50.40RH;
<DevID>10 T30.48C 50.60RH;
<DevID>10 T30.45C 50.80RH;
<DevID>10 T30.44C 51.40RH;
<DevID>10 T30.44C 52.20RH;
<DevID>10 T30.42C 51.50RH;
<DevID>10 T30.41C 53.00RH;
<DevID>10 T30.40C 52.20RH;

<DevID>10 T30.43C 52.40RH;
<DevID>10 T30.45C 52.40RH;

### 2.8.2 Multi node data log

Following sample log file shows the data file format of the SD card when multiple sensor nodes exist in the system. The Router station operates just as a router device in this case and no sensor connected to the router. Note that the node ID varies from entry to entry and are 3 digit numbers. It indicates the data comes from end device (3rd level device) which is a sensor module in the network.

++++++++++++++ New Session Started ++++++++++++++
<DevID>100 T20.50C 50.45RH;
<DevID>101 T32.52C 40.50RH;
<DevID>103 T26.51C 55.20RH;
<DevID>102 T35.51C 61.40RH;
<DevID>101 T32.52C 42.40RH;
<DevID>104 T19.58C 52.30RH;
<DevID>105 T30.62C 52.60RH;
<DevID>102 T35.63C 61.00RH;
<DevID>106 T40.64C 49.40RH;
<DevID>104 T19.64C 49.60RH;
<DevID>101 T32.58C 49.70RH;
<DevID>102 T35.56C 60.20RH;
<DevID>106 T40.52C 50.40RH;
<DevID>104 T19.48C 50.60RH;
<DevID>101 T32.45C 50.50RH;
<DevID>106 T40.44C 51.40RH;
<DevID>102 T30.44C 62.20RH;
<DevID>105 T30.42C 51.50RH;
<DevID>101 T32.41C 43.20RH;
<DevID>104 T19.40C 52.10RH;

<DevID>103 T26.43C 55.40RH;
<DevID>105 T30.45C 52.50RH;
<DevID>100 T20.60C 50.45RH;

Above sample log file contains data received from node ID100,101,102,103,104,105and106. Further all those devices can be known as end devices, since the ID has 3 digits

### 2.9 Communication and security

#### 2.9.1 XBee communication basics

ZigBee is a specification for a suite of high level communication protocols used to create personal area networks built from small, low-power digital radios. ZigBee is based on an IEEE 802.15 standard. Though low-powered, ZigBee devices can transmit data over long distances by passing data through intermediate devices to reach more distant ones, creating a mesh network; i.e., a network with no centralized control or high-power transmitter/receiver able to reach all of the networked devices. The decentralized nature of such wireless ad hoc networks makes them suitable for applications where a central node can't be relied upon.

ZigBee is used in applications that require only a low data rate, long battery life, and secure networking. ZigBee has a defined rate of 250 kbit/s, best suited for periodic or intermittent data or a single signal transmission from a sensor or input device. Applications include wireless light switches, electrical meters with in-home-displays, traffic management systems, and other consumer and industrial equipment that requires short-range wireless transfer of data at relatively low rates. The technology defined by the ZigBee specification is intended to be simpler and less expensive than other WPANs, such as Bluetooth or Wi-Fi.[1]

### 2.9.2 Security

As one of its defining features, ZigBee provides facilities for carrying out secure communications, protecting establishment and transport of cryptographic keys, ciphering frames and controlling devices. It builds on the basic security framework defined in IEEE 802.15.4. This part of the architecture relies on the correct management of symmetric keys and the correct implementation of methods and security policies.[2]

ZigBee networks are secured by 128 bit symmetric encryption keys. In home automation applications, transmission distances range from 10 to 100 meters line-of-sight, depending on power output and environmental characteristics.

### 2.10 Network coverage and range test

X-CTU can be used for testing the range as same as for the configuration of XBee. It provides a nice tool to test the range of the XBee modules and the quality of the link. Open the X-CTU application and click on the “Range test” tab. This can be used as the range tester in the XBee modules.

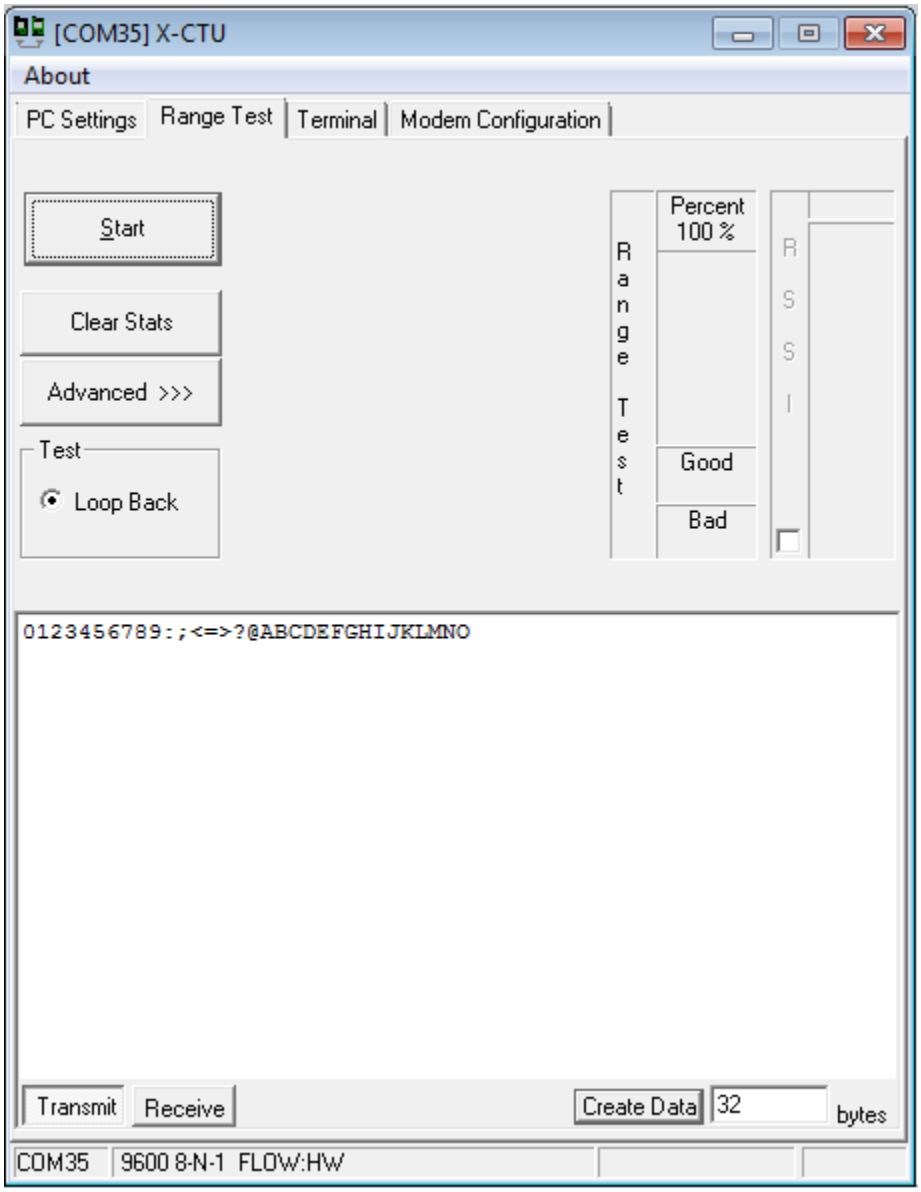


**Fig 2.19 X-CTU range test**

# 3 Conclusion

Multi node data routing and data logger system have already implemented many advanced features and it's fully functional. However there are certain areas still to be developed. The following simple comparison is about the system pros and cons of the existing system architecture.

**Pros**

1) Multiple sensor node support.
2) Long data transmission range.
3) Basic packet structure validation at two stages.
4) Basic MAC addresses validation before establish the communication.
5) Data logging capability to removable SD card.
6) Minimum device boot up time (< 2Sec)
7) Capability to capture the original data source Device ID (Sensor node ID store with data string)
8) Up to 32GB SD card support
9) XBee® 802.15.4Communication standard.
10) Temperature and humidity measurement accuracy up to (+/- 1 degree C and +/- 3.5%RH)

**Cons**

1) Time stamp is not stored. (This feature can be easily implemented by adding RTC module at the data logger station circuit).
2) Full data routing path is not transparent. (At the moment the each data entry only contains the original senor node ID. It doesn't include the data router station ID. This feature is really necessary when multiple data routers are exist in the same domain)
3) Failed data recall option is not implemented. (If the data packet validation is failed, that data packet can be recalled from the sensor node by simply modifying the data router algorithm)
4) Power consumption can further be optimized by simply modifying the sensor module algorithm.

## 4 References

[1] Available at: http://en.wikipedia.org/wiki/ZigBee

[2] Available at: http://en.wikipedia.org/wiki/ZigBee#ZigBee_PRO

[3] Available at: https://www.digi.com/technology/rf-articles/wireless-zigbee

[4] Available at: http://arduino.cc/